Analog Descriptor Add-ons

Remembering special tidbits

Tue, Feb 25, 1997

When it is necessary to keep special information in the local database that relates to only a few channels, a means of storing it would be desirable. This note explores possibilities for this.

Using the name table

The name table is designed to provide a quick way to search for info according to a unique key. It has been used so far in name searching, for both 6-character names used locally in the Classic protocol and for 16-character names used in the D0 detector control system. The scheme depends upon the use of unique keys. Multiple types of keys may be used, because in each case a type code is specified along with the name. The "name," or whatever unique key is used, must reside within a system table of some kind. The data that is stored in the "name table" is only the entry number of the system table that is referenced by the type code. For 6-character names, for example, the data that is stored in the name table entry is the channel#, along with a pointer to the name as it resides in a field of the analog descriptor table.

In this case, the "name" might be a channel#, as those are unique within a system by definition. A system table would have to exist to hold the information that is special for a few channels. The data word in the name table entry would be the entry number of that new system table. The pointer would have to point to the channel# that must be stored in that entry. So, given a channel#, one can quickly come up with the corresponding entry# in the new system table, if there is one.

A new listype# could be defined, along with new read routines and set routines, so that read/write access to the special information can be supported. Some means of deleting the extra info, and thus opening up a spare slot in the new table, must also be provided.

Reference from descriptor

The spare byte in the analog descriptor could be used to index into an up-to-256-entry table that could contain the extra info. It is quick to find the special info, given a channel#, but one is limited to 256 entries.

Increase size of descriptor

Although this is the easiest path, it requires more memory than may be convenient to relinquish. The 162 boards have only 512K of non-volatile memory. For 1024 channels, we use 64K of this memory for the current descriptor size. With 192K of downloaded program memory, only 256K remains. If the #channels is limited to 1024, then we could double the size of a descriptor entry with no trouble. Not having the special D0 tables saves another 64K. Not expanding the ADATA and BDESC tables

In summary, limiting support to 1K channels and 1K bits, with 128 bytes/analog descriptor table entry, we have:

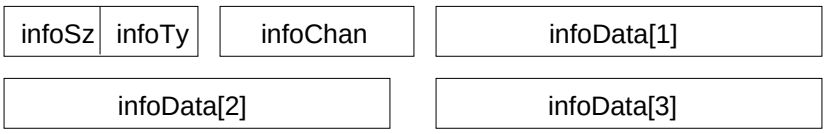
Download program area	192K
ADESC	128K
ADATA	16K
BDESC	16K
Miscellaneous	64K
spare	96K
Total	512K

One could double the size of an ADATA and/or BDESC entry and still have 64K available.

For the case of 133-based systems, which are all those in Linac, the situation is different, although there is 1024K of non-volatile memory available. With the ADESC table set at 150000, and with 2K channels of 128 bytes each, we would extend the ADESC table to 190000, where the binary tables begin.

New CINFO system table

Another implementation of channel-related extra information uses another system table, #25. Each entry contains the channel# key. An entry could be variable length and of different types. Here is a proposed layout:



Each entry is composed of a size byte, a type byte, a channel# word, and arbitrary information. An entry's size must be a multiple of 8 bytes. The example above shows an entry with a size of 16 bytes.

As an example of the use of such an auxiliary table, consider the need for information about channels that have swift digitizer support in Booster IRMs. Up to 8 channels can have such support, but which channels they are is arbitrary. The FTPMAN support must, given a channel#, determine whether or not swift digitizer support exists for that channel. The CINFO scheme can do this. A new library routine can assist in searching this small table.

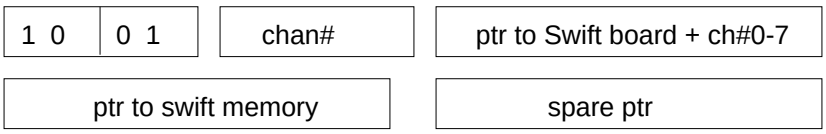
```
Function CINFOEntry(typ, chan: Integer): CINFOPtr;
```

In the case that there is no CINFO entry available for the given channel, NIL is returned. Let the type parameter be 1 for swift digitizer information. When FTPMAN gets a request for swift digitizer data for a given channel, this routine can answer the

question of whether that channel is connected to a swift digitizer channel. For example, the CINFO entry might have this form:

The ptr to the swift digitizer board gives access to the registers for controlling the digitizer. The low 3 bits can contain which of the 8 possible digitizer channels is used. To find the memory occupied by the resulting digitized waveform, one must read a register from the IPIC chip, whose address is fixed, on the 162 board. The register that must be read depends upon which of the four IP board sockets is used. The base addresses of the IP board sockets on the 162 cpu board are FFF58x00, where x ranges from 0–3 for board sockets a, b, c, d. So the board socket index can be obtained from the base address that is contained in the table, such as FFF5820y for board socket 2, or b, with y in the range 0–7 to signify which digitizer channel is used.

In case an separate IP carrier board is used, and the IPIC chip is not used, more information is needed to find the memory address of the waveform. To cover this case, the size should be larger than 8 bytes. Here is a possible layout:



Here is a routine that can return the ptr to a swift digitizer waveform memory:

```
Function SwiftMem(infoP: CINFOPtr): SwiftMPtr;
```

Given a pointer to the CINFO table entry that was returned by CINFOEntry, SwiftMem uses the previous logic to find the base address for the given waveform's memory. It checks for both the short and long cases. It returns NIL if an error.

One can also, in a similar way, design a scheme to help with quick digitizer support.